Spring 5-1-1995

Packing Algorithms for Arborescences (and Spanning Trees) in Capacitated Graphs ; CU-CS-773-95

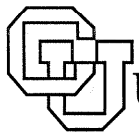
Harold N. Gabow
University of Colorado Boulder

Manu
University of Colorado Boulder

Packing Algorithms for Arborescences (and Spanning Trees) in Capacitated Graphs *

Harold N. Gabow
K. S. Manu

CU-CS-773-95



University of Colorado at Boulder
DEPARTMENT OF COMPUTER SCIENCE

* Research supported in part by NSF Grant No. CCR-9215199

Packing Algorithms for Arborescences (and Spanning Trees) in Capacitated Graphs

Harold N. Gabow* and K.S. Manu*

Department of Computer Science, University of Colorado at Boulder, Boulder, CO 80309

Abstract. In a digraph with real-valued edge capacities, we pack the greatest number of arborescences in time $O(n^3 m \log(n^2/m))$; the packing uses at most m distinct arborescences. Here n and m denote the number of vertices and edges in the given graph, respectively. Similar results hold for integral packing: we pack the greatest number of arborescences in time $O(\min\{n, \log(nN)\}n^2 m \log(n^2/m))$ using at most $m + n - 2$ distinct arborescences; here N denotes the largest (integral) capacity of an edge. These results improve all previous strong- and weak-polynomial bounds for capacitated digraphs. The algorithm extends to related problems, including packing spanning trees in an undirected capacitated graph, and covering such graphs by forests. The algorithm provides a new proof of Edmonds' theorem for arborescence packing, for both integral and real capacities. The algorithm works by maintaining a certain laminar family of sets.

1 Introduction

Packing arborescences and spanning trees are among the most basic packing problems. They probably provide the best-known examples of the so-called strong integrality and integer rounding properties of packing, respectively [Tro]. Many efficient algorithms for both problems have been presented for (unit capacity) graphs. This paper presents the most efficient algorithms known for graphs with arbitrary capacity functions. We give both strong- and weak-polynomial bounds.

We now state our results and compare them to previous work. In the time bounds throughout this paper, n and m denote the number of vertices and edges of the given graph, respectively. The parameter N represents the largest capacity. A bound involving N assumes that all given capacities are integers (between 0 and N) and so is a weak-polynomial bound.

The bulk of this paper is devoted to packing arborescences (more precisely a-arborescences). Edmonds proved the strong integrality property for arborescence packing [E72] (stated precisely at the end of this section). Other proofs are in [F79, G91, L, Mad, TL]. Algorithms for packing arborescences in (unit capacity) graphs are presented in [Ta, S, TL]. The most efficient algorithm for general unit capacity graphs finds k edge-disjoint arborescences in time $O((kn)^2)$ [G91]. Polynomial-time algorithms for packing arborescences in capacitated graphs are given in [GLS, Mad].

* Research supported in part by NSF Grant No. CCR-9215199.

We present two related algorithms for packing arborescences in capacitated graphs. The first finds a maximum packing in a digraph with arbitrary real-valued capacities. We call this a fractional packing – a given arborescence can be used α times for any nonnegative real value α. (This type of packing is needed for the problems of [PW], see below.) The second algorithm finds a maximum integral packing in a digraph with integral capacities. The two algorithms provide independent proofs of the real and integral cases of Edmonds' theorem respectively [E72]. The proof differs from previous ones in that it is based on maintaining a laminar family of sets.

Our fractional packing algorithm runs in time $O(n^3 m \log(n^2/m))$. Our integral packing algorithm has a slightly better time bound, $O(\min\{n, \log(nN)\}n^2 m \log(n^2/m))$. These are the best-known bounds for both versions of the problem. Our algorithms find packings with essentially the fewest possible number of distinct arborescences: we use at most m distinct arborescences for fractional packing and $m + n - 2$ distinct arborescences for integral packing. (We note that Mader's approach to packing by splitting off edges [Mad] can be implemented efficiently using the ideas of this paper. However the best bound on the number of distinct arborescences that we can prove for this approach is $O(n^2)$.)

We present four applications of the packing algorithms. Each runs in the above time bounds (for the fractional and integral cases) and finds a packing or covering, the number of whose distinct members is close to the above number of distinct arborescences. Two applications are for capacitated digraphs. The first is finding a minimum (fractional or integral) covering of a digraph by branchings. The second is finding a maximum root-constrained packing of arborescences. (In this problem lower and upper bounds are given for the number of times each vertex can be the root of an arborescence.) Minimax formulas for root-constrained packing are given in [F78, Mao].

The two other applications are for capacitated undirected graphs. They are finding a maximum packing of spanning trees, and finding a minimum covering by forests. Edmonds gave the first polynomial-time algorithms for these problems, establishing the integer rounding property [E65]. Efficient algorithms are given in [G95, GW, I, RT] for (unit capacity) graphs and [GLS, PW] for capacitated graphs. The most efficient algorithms for packing and covering in capacitated graphs are due to Trubin [Tru91] running in the time for $O(n^3)$ network flow computations. We improve this by a factor of $\max\{n, n^2/\log(nN)\}$. Our algorithms for these undirected problems are based on the reduction in [G95] of undirected packing and covering to the directed case.

The paper is organized as follows. Section 2 discusses the greedy algorithm for fractional arborescence packing. It proves the fractional case of Edmonds' arborescence packing theorem. Section 3 provides the last part of our fractional packing algorithm, an efficient method to compute the ca-

pacity of an arborescence. Section 4 presents an integral packing algorithm that finds arborescences one-by-one. It proves the integral case of Edmonds' theorem. Section 5 extends this to our efficient integral packing algorithm. Section 6 presents the applications to digraph covering and undirected packing and covering. Section 7 discusses root-constrained packing. The rest of this section gives notation and definitions.

$\mathbf{R}^+$ denotes the set of positive real numbers.

Consider a universe V containing elements s, t and subsets S, T. $\overline{S}$ denotes the complement $V - S$. We often denote singleton sets by omitting set braces, e.g., $S \cup s$. An *$s\bar{t}$-set* contains s but not t; a *$\bar{t}$-set* is a nonempty set not containing t. Sets S and T are *intersecting* if $S \cap T$, $S - T$ and $T - S$ are all nonempty. A family of subsets of V is *laminar* if every two sets in the family are either disjoint or one contains the other.

Consider a directed or undirected graph $G = (V, E)$. If A is a set of edges $V(A)$ denotes the set of all vertices on these edges. A *capacity function* is a mapping $c : E \to \mathbf{R}^+$. By definition any edge has positive capacity. If an edge has zero capacity we always delete it from E.

An edge uv *enters* any $v\bar{u}$-set and *leaves* any $u\bar{v}$-set. For a set of vertices W, the *in-degree* $\rho_G(W)$ equals the total capacity of all edges entering W. Any graph in this paper will have a capacity function, so this notation always makes sense. On the other hand if A is a set of edges, we shall implicitly assume that each edge has unit capacity in A. Thus $\rho_A(W)$ denotes the number of edges entering W. This convention will be used frequently, but to prevent confusion the set of edges will always be named A when the convention is used.

The following submodular-type identity is well-known and easy to verify. Let $d_G(S,T)$ denote the total capacity of all edges having one end in $S - T$ and the other in $T - S$. Then for any sets $S, T \subseteq V$, $\rho(S) + \rho(T) = \rho(S \cup T) + \rho(S \cap T) + d(S,T)$. Here we abbreviate ρ_G to ρ and d_G to d.

A *branching* in a digraph is a set of edges A such that any vertex v has $\rho_A(v) \leq 1$, and ignoring edge directions A is a forest. Branching A is an *arborescence* if $V(A)$ contains a unique vertex a with $\rho_A(a) = 0$; in this case A is an *a-arborescence*. A is *spanning* if $V(A) = V$.

Consider a digraph $G = (V, E)$ with distinguished vertex a, and a capacity function $c : E \to \mathbf{R}^+$. A *fractional packing of a-arborescences* is a family $\mathcal{A}$ of spanning a-arborescences A, each with an associated multiplicity $\alpha(A) \in \mathbf{R}^+$ such that for any edge e, $\sum\{\alpha(A) : e \in A \in \mathcal{A}\} \leq c(e)$. The *total multiplicity* of $\mathcal{A}$ is $\sum\{\alpha(A) : A \in \mathcal{A}\}$. An *integral packing* has all multiplicities $\alpha(A)$ integral. A *maximum packing* has maximum total multiplicity. The main problem of this paper is to find maximum fractional and integral packings of a-arborescences.

Let $\lambda(G)$ denote the minimum in-degree of an $\overline{a}$-set, i.e., $\lambda(G) = \min\{\rho_G(X) : \emptyset \neq X \subseteq V - a\}$. Edmonds showed that in any digraph with integral capacities, a maximum integral packing $\mathcal{A}$ has total multiplicity $\lambda(G)$ [E72]. Throughout this paper we always define $\lambda(G)$ using the vertex named a.

2 Fractional Packing

This section discusses the greedy approach to fractional packing. It proves the fractional case of Edmonds' arborescence packing theorem and gives our efficient algorithm for fractional packing.

Consider an arbitrary set of edges A (the following notions will be used mainly when A is an arborescence). Define

$$\mu(A) = \min\{c(e) : e \in A\}.$$

For any α, $0 \leq \alpha \leq \mu(A)$, let $G - \alpha A$ denote the graph G with the given capacity function c decreased by α on the edges of A. (As mentioned in Section 1, in $G - \alpha A$ we delete any edge that has zero capacity.) As an example when all capacities are one, $G - A$ is the graph with the edges A removed.

The *capacity of* A is the largest value $\alpha \leq \mu(A)$ having $\lambda(G - \alpha A) = \lambda(G) - \alpha$. The capacity always exists, although it may be 0. We sometimes use the notation $\alpha(A)$ to denote the capacity of A.

Section 3 discusses this notion of capacity in detail. For now the following characterization will suffice. If a set of edges A has $\mu(A) > 0$ yet $\alpha(A) = 0$ then some $\overline{a}$-set U has

$$\rho_G(U) = \lambda(G) \text{ and } \rho_A(U) > 1.$$

In proof suppose the opposite, i.e., every $\overline{a}$-set U has $\rho_G(U) > \lambda(G)$ or $\rho_A(U) = 1$. Then $\rho_{G-\alpha A}(U) = \rho_G(U) - \alpha\rho_A(U) \geq \lambda(G) - \alpha$ holds for small enough positive α in the first case and for any α in the second case (since $\rho_G(U) \geq \lambda(G)$). Thus $\mu(A) > 0$ as desired.

Consider the following *greedy algorithm* for fractional packing: Choose any spanning a-arborescence A of G that has positive capacity α. Place A in the packing with multiplicity α and replace G by $G - \alpha A$. If $\lambda(G)$ is still positive then find the rest of the packing recursively.

We will prove that the greedy algorithm works (in particular A exists) and does essentially the best job possible: It constructs a maximum packing, using at most m distinct arborescences. The main ingredient of the proof is a laminar family of sets $\mathcal{F}$. We will use different versions of $\mathcal{F}$ for fractional and integral packing. We begin with remarks about $\mathcal{F}$ that hold for both versions.

$\mathcal{F}$ is a laminar family of nonsingleton $\overline{a}$-sets. Thus $\mathcal{F}$ contains at most $n-2$ sets (since $\mathcal{F}$ plus all singleton $\overline{a}$-sets is a laminar family on $n-1$ elements). We always initialize $\mathcal{F}$ to $\emptyset$. We only change $\mathcal{F}$ using the following operation, defined for an $\overline{a}$-set U. To *uncross* $\mathcal{F} \cup \{U\}$ means to execute the following procedure:

$YI \leftarrow$ the family of maximal sets of $\mathcal{F}$ that are intersecting with U;
$Z \leftarrow U \cup \bigcup\{Y : Y \in YI\}$;
add Z to $\mathcal{F}$;

The family YI can be empty. If U is a nonsingleton set (as will always be the case) then so is Z. The new family $\mathcal{F}$ is laminar. (In proof let $\mathcal{F}$ denote the family before uncrossing. There is a unique minimal set $X \in \mathcal{F} \cup \{V\}$ that contains U. Any set of $\mathcal{F}$ contained in X is either disjoint from Z or contained in U or contained in some $Y \in YI$. Hence $\mathcal{F} \cup \{Z\}$ is laminar.)

The uncrossing operation may not change $\mathcal{F}$ since $\mathcal{F}$ may have already contained Z. We will always prove that when we uncross, $\mathcal{F}$ actually gets enlarged.

Now we return to the specific case of fractional packing. The complete specification of $\mathcal{F}$ is that it consists of nonsingleton $\overline{a}$-sets X that satisfy

$$\rho_G(X) = \lambda(G).$$

Recall that our algorithm changes the graph G as it constructs the packing; in this equation G refers to the current graph. Let us summarize the properties of various sets of the uncrossing operation in fractional packing.

Lemma 2.1. *Suppose we uncross $\mathcal{F} \cup \{U\}$ for an $\overline{a}$-set U that has $\rho_G(U) = \lambda(G)$. Let Y and Y' be any sets in YI. Then $\rho_G(Z) = \rho_G(Y \cap U) = \lambda(G)$ and $d_G(Y,U) = d_G(Y-U, Y'-U) = 0$.*

Proof. In this argument we omit the subscript from ρ_G and d_G. The submodular identity $\rho(Y) + \rho(U) = \rho(Y \cap U) + \rho(Y \cup U) + d(Y,U)$ implies $\rho(Y \cap U) = \rho(Y \cup U) = \lambda(G)$ and $d(Y,U) = 0$. A similar argument applied to $Y \cup U$ and $Y' \cup U$ shows $d(Y \cup U, Y' \cup U) = d(Y-U, Y'-U) = 0$. Finally repeated applications of the argument, using $Z = \bigcup\{Y \cup U : Y \in YI \cup \{\emptyset\}\}$, show $\rho(Z) = \lambda(G)$. □

It is convenient to work with a more detailed version of the greedy algorithm. (We relate this version to the original greedy algorithm after Theorem 2.1.) The following version of the greedy algorithm uses $\mathcal{F}$ to find an arborescence A with nonnegative (possibly 0) capacity.

procedure *fractional_pack*; **begin**

$\mathcal{F} \leftarrow \emptyset$;

while $\lambda(G) > 0$ **do begin**

$A \leftarrow$ a spanning a-arborescence of G with $\rho_A(X) = 1$ for every $X \in \mathcal{F}$;

$\alpha \leftarrow$ the capacity of A; /* possibly $\alpha = 0$ */

$G \leftarrow G - \alpha A$;

if $\mu(A) > 0$ **then begin** /* no edge was voided */

$U \leftarrow$ a minimal $\overline{a}$-set with $\rho_G(U) = \lambda(G)$ and $\rho_A(U) > 1$;

uncross $\mathcal{F} \cup \{U\}$; **end**; **end**; **end**;

This algorithm implicitly assumes that each arborescence A with positive capacity α gets added to the packing with multiplicity α.

The existence of the set U was shown above (since A has capacity 0 after αA is deleted). U is not a singleton set since $\rho_A(U) > 1$. To show *fractional_pack* is well-defined we must prove that the spanning a-arborescence A exists. We begin with a key fact.

Fix any set $X \in \mathcal{F} \cup \{V\}$. Define G_X as the subgraph of G induced by X, with every maximal set of $\mathcal{F}$ that is properly contained in X contracted. Choose a vertex b in G_X as follows: If $X = V$ then $b = a$. If $X \neq V$ then choose b so that some edge of G entering X is directed to b.

Lemma 2.2. *G_X has a spanning b-arborescence.*

Proof. It suffices to show that for every $\overline{b}$-set S in G_X, some edge of G goes from $X - S$ to S. If $X = V$ this follows from $\lambda(G) > 0$. Suppose $X \neq V$. The edge defining b contributes to $\rho_G(X)$ but not $\rho_G(S)$. Yet $\rho_G(X) = \lambda(G) \leq \rho_G(S)$. Thus some edge contributes to $\rho_G(S)$ but not $\rho_G(X)$, i.e., it enters S but not X. This is the desired edge. □

Now we show the arborescence A of *fractional_pack* exists by giving an algorithm to construct it. Initialize A to a spanning a-arborescence of G_V. Then repeat the following step until A spans G: Choose a set $X \in \mathcal{F}$ such that A contains an edge directed to a vertex b of G_X but A does not span G_X. Enlarge A by adding a spanning b-arborescence of G_X.

Lemma 2.2 shows the spanning a- and b-arborescences exist. A simple induction shows that A is an arborescence. Each $X \in \mathcal{F}$ has $\rho_A(X) = 1$ (by construction). Thus the algorithm constructs A as specified in *fractional_pack*.

Theorem 2.1. *Procedure* fractional_pack *finds a fractional packing with total multiplicity* $\lambda(G)$*, containing at most* m *distinct arborescences.*

Proof. The definition of capacity implies that any iteration decreases $\lambda(G)$ by α and increases the multiplicity of the packing by α. Thus if *fractional_pack* halts (necessarily with $\lambda(G) = 0$) the packing has the desired multiplicity. So to prove the theorem we need only show that *fractional_pack* halts having found at most m distinct arborescences.

We will show that the family $\mathcal{F}$ gets enlarged by any uncrossing operation. This implies the theorem. To see this note that $\mathcal{F}$ gets enlarged at most $n-2$ times. An iteration that does not enlarge $\mathcal{F}$ voids an edge of G when αA is deleted. Before the last iteration this occurs at most $m-n+1$ times (since the last arborescence contains $n-1$ edges). Thus the number of iterations, and hence the number of distinct arborescences, is at most $(m-n+1)+1+(n-2) = m$.

To complete the proof consider an iteration that does an uncrossing operation. We wish to show that uncrossing $\mathcal{F} \cup \{U\}$ enlarges $\mathcal{F}$, i.e., the set Z of the uncrossing procedure was not already in $\mathcal{F}$. We claim $\rho_A(Z) \geq \rho_A(U)$. Since $\rho_A(U) > 1$ (by the choice of U) the claim implies $\rho_A(Z) > 1$. Hence $Z \notin \mathcal{F}$ before uncrossing (by the choice of A). Thus the claim suffices to prove the theorem.

To prove the claim we first note two relations. Let the iteration end with graph G. Any set $Y \in YI$ has

$$\rho_A(Y \cap U) = 1. \tag{1}$$

This follows since Lemma 2.1 shows $\rho_G(Y \cap U) = \lambda(G)$, so if (1) fails then $Y \cap U$ contradicts the minimality of U. (This is the only place the proof uses the minimality of U. This fact will be used below.)

The second relation is that any sets $Y, Y' \in YI$ have

$$d_A(Y, U) = d_A(Y - U, Y' - U) = 0. \tag{2}$$

In proof, Lemma 2.1 shows (2) for the function d_G. Since an uncrossing operation is done, no edge of A was voided, i.e., every edge of A has positive capacity in G. Thus (2) holds for d_A.

To prove the claim it suffices to show that any edge $e \in A$ entering U corresponds to a unique edge of A entering Z. If e enters both U and Z it corresponds to itself. Suppose e enters U but not Z. Thus for some $Y \in YI$, e leaves $Y - U$. Now (2) shows e goes from $Y - U$ to $Y \cap U$. Thus (1) shows e is unique for the set Y. Since some edge $f \in A$ enters Y, (1) shows f goes from $\overline{Y}$ to $Y - U$. Now (2) shows f goes from $\overline{Z}$ to $Y - U$. Since f enters Z we can let e correspond to f. It is clear that e is the only edge corresponding to f. □

The theorem implies our claim for the greedy algorithm, i.e., it finds a maximum packing, and this packing contains at most m distinct arborescences A. This follows because any arborescence A chosen by the greedy algorithm can be chosen by *fractional_pack*, so the two algorithms construct the same packing.

The theorem also gives an independent proof of the real-valued case of Edmonds' theorem [E72]: Any digraph with real capacities has a fractional packing of a-arborescences of total multiplicity $\lambda(G)$.

To implement *fractional_pack* efficiently Section 3 gives an algorithm to compute the capacity of A. The algorithm also computes a set $\widehat{U}$ that satisfies all the properties of U in *fractional_pack* except minimality. To use $\widehat{U}$ instead of U in *fractional_pack* replace the line that computes U by the following two lines:

$\widehat{U} \leftarrow$ an $\overline{a}$-set with $\rho_G(\widehat{U}) = \lambda(G)$ and $\rho_A(\widehat{U}) > 1$;
$U \leftarrow \widehat{U} \cap X$ for X a minimal set of $\mathcal{F} \cup \{V\}$ having $\rho_A(\widehat{U} \cap X) > 1$;

Let us show that Theorem 2.1 holds for this implementation of *fractional_pack*. It suffices to show that the set U constructed above has all the necessary properties of U in the original algorithm. The first property is that $\rho_G(U) = \lambda(G)$. This translates to $\rho_G(\widehat{U} \cap X) = \lambda(G)$, which is true by the submodularity argument of Lemma 2.1. The second property of U is $\rho_A(U) > 1$. This translates to $\rho_A(\widehat{U} \cap X) > 1$, which holds by definition. The final property of U is its minimality, but recall this is only used to establish (1). Since in (1) Y is intersecting with $U = \widehat{U} \cap X$, $Y \subseteq X$. Thus $Y \cap U = Y \cap \widehat{U}$. The minimality of X shows $\rho_A(Y \cap \widehat{U}) = 1$. This implies (1).

Now we show that assuming the results of the next section, this implementation of *fractional_pack* achieves our time bound for fractional packing.

Corollary 2.1. *Procedure* fractional_pack *as implemented above runs in time* $O(n^3 m \log(n^2/m))$.

Proof. We find the arborescence A in time $O(m)$ using the recursive procedure given above. Suppose $\alpha = \mu(A)$ (i.e., deleting αA voids an edge). In this case Lemma 3.1 shows the capacity is found in time $O(nm \log(n^2/m))$. Since there are $\leq m \leq n^2$ iterations of *fractional_pack*, the total time for these iterations is within the desired time bound.

Next suppose $\alpha < \mu(A)$. In this case the capacity is found in time $O(n^2 m \log(n^2/m))$, as well as the set $\widehat{U}$ of the implementation (by Lemma 3.1 and the remark following it). Sets X and U are found in a straightforward way in time $O(n^2)$. (Find X by testing each set of $\mathcal{F}$.) The uncrossing

operation can also be done in this time bound. There are at most n iterations with $\alpha < \mu(A)$ since each enlarges $\mathcal{F}$. This gives the desired time bound. □

3 Capacity Computation

Let A be an arbitrary set of edges. This section computes the capacity $\alpha(A)$ using a version of Newton's method for finding a root.

We begin by reviewing the definition of capacity. An equivalent version of the definition is that the capacity is the largest value $\alpha \leq \mu(A)$ such that for each $\overline{a}$-set U, $\rho_{G-\alpha A}(U) = \rho_G(U) - \alpha\rho_A(U) \geq \lambda(G) - \alpha$. If $\rho_A(U) = 1$ the desired inequality holds for any α, since $\rho_G(U) \geq \lambda(G)$. If $\rho_A(U) > 1$ then the desired inequality holds if and only if $\alpha \leq \alpha_U$, where we define

$$\alpha_U = \frac{\rho_G(U) - \lambda(G)}{\rho_A(U) - 1}.$$

It is possible that $\alpha_U = 0$.

This characterization of α motivates the following algorithm to compute the capacity.

$\alpha \leftarrow \mu(A)$;
while $\lambda(G - \alpha A) < \lambda(G) - \alpha$ **do begin**
    $U \leftarrow$ an $\overline{a}$-set of minimum in-degree in $G - \alpha A$;
    $\alpha \leftarrow \alpha_U$; **end**;

To verify this algorithm is correct we must check two facts. First the algorithm never divides by 0 in computing α_U, since it is easy to see that $\rho_A(U) > 1$. The second fact is that the algorithm halts. We will prove this by showing there are at most $|A| + 1$ iterations.

First observe that the sequence of values of α is strictly decreasing. In proof, recall that U has in-degree $\geq \lambda(G) - \alpha$ exactly when $\alpha \leq \alpha_U$. Thus each iteration changes α from a value larger than α_U to α_U.

Next we claim that if an iteration computes the set U, then any $\overline{a}$-set X with $\rho_A(X) \geq \rho_A(U)$ has $\alpha_X \geq \alpha_U$. Observe that the claim implies the desired bound: It implies the sequence $\rho_A(U)$ is strictly decreasing. Since $0 \leq \rho_A(U) \leq |A|$ the desired bound of at most $|A| + 1$ iterations follows.

To prove the claim let α equal its value when the iteration begins and U is computed. The definition of U implies $\rho_G(X) - \alpha\rho_A(X) \geq \rho_G(U) - \alpha\rho_A(U)$. Using $\alpha_U < \alpha$ and the assumption $\rho_A(X) \geq \rho_A(U)$ gives $\rho_G(X) - \alpha_U\rho_A(X) \geq \rho_G(U) - \alpha_U\rho_A(U) = \lambda(G) - \alpha_U$. This inequality implies the claim $\alpha_X \geq \alpha_U$.

Lemma 3.1. *The capacity $\alpha(A)$ of an arborescence A can be computed in time $O(\min\{n, \log(nN)\}$ $nm\log(n^2/m))$. If $\alpha(A) = \mu(A)$ the time is $O(nm\log(n^2/m))$.*

Proof. We check the termination condition of the loop and find the set U using the global minimum cut algorithm of Hao and Orlin [HO]. This gives time $O(nm\log(n^2/m))$ for one iteration.

Now the general time bound follows from the fact that the number of iterations is $O(\min\{n, \log(nN)\})$. The bound n was proved above (since $|A| = n-1$). The bound $O(\log(nN))$ (in the case of integral capacities) holds for any implementation of the Newton-Dinkelbach method for finding the minimum of a function $u(x)/v(x)$ [R]. It is easy to show that our algorithm is an implementation of this method. (Alternatively we can achieve the bound $O(\log(nN))$ by doing a binary search for the capacity.)

If $\alpha(A) = \mu(A)$ there is only one iteration. This gives the second time bound. □

We close with two remarks. If $\alpha(A) < \mu(A)$, the last set U has $\rho_A(U) > 1$ and $\alpha(A) = \alpha_U$; writing $\alpha = \alpha(A)$ we have $\rho_{G-\alpha A}(U) = \lambda(G) - \alpha = \lambda(G - \alpha A)$. This set U can be returned as the set $\widehat{U}$ of procedure *fractional_pack*.

The integral packing algorithm of Section 5 computes the floor of the capacity. It is easy to see that our algorithm works if we change the last line to $\alpha \leftarrow \lfloor \alpha_U \rfloor$. Lemma 3.1 continues to hold. The advantage of this version is that all computations preserve integrality.

4 One-by-one Packing

This section presents an integral packing algorithm. It finds each arborescence (of multiplicity one) individually. The algorithm gives an independent proof of Edmonds' theorem for integral arborescence packing, and is used in our general integral packing algorithm.

Consider a digraph $G = (V, E)$ with distinguished vertex a. G may have an integral capacity function, in which case we consider each edge to have multiplicity equal to its capacity. Throughout this section we write $\lambda = \lambda(G)$. We will grow an a-arborescence A of the packing edge-by-edge. Initially A is empty, and eventually it becomes a spanning arborescence. We maintain A to always satisfy the invariant,

$$\lambda(G - A) \geq \lambda - 1. \tag{3}$$

When a spanning a-arborescence A satisfies (3) then (3) holds with equality. In that case we add A to the packing and recursively pack the graph $G - A$.

To guide the construction of A we maintain a laminar family $\mathcal{F}$ of $\overline{a}$-sets X, each of which has

$$\rho_{G-A}(X) = \lambda - 1 \text{ or } \rho_G(X) = \lambda. \tag{4}$$

Note that a set $X \in \mathcal{F}$ that does not satisfy the first condition has $\rho_A(X) = 0$ (since (3) implies $\rho_{G-A}(X) \geq \lambda$, which with $\rho_G(X) = \lambda$ implies the desired conclusion).

When the packing algorithm begins to grow the very first arborescence, $\mathcal{F}$ is initialized to $\emptyset$. Thereafter the only operation on $\mathcal{F}$ is to uncross $\mathcal{F} \cup \{U\}$, where U is a (nonsingleton) $\overline{a}$-set with $\rho_{G-A}(U) = \lambda - 1$. Now we summarize the properties of various sets when we uncross $\mathcal{F} \cup \{U\}$.

Lemma 4.1. *Suppose we uncross $\mathcal{F} \cup \{U\}$ for an $\overline{a}$-set U that has $\rho_{G-A}(U) = \lambda - 1$. Then $\rho_{G-A}(Z) = \lambda - 1$. For any set $Y \in YI$, $\rho_{G-A}(Y) = \rho_{G-A}(Y \cap U)$ and $d_{G-A}(Y, U) = 0$.*

Proof. In this argument ρ and d refer to the graph $G - A$. We first prove the two claims involving Y, as well as the relation $\rho(Y \cup U) = \lambda - 1$. Consider the submodular identity $\rho(Y) + \rho(U) = \rho(Y \cap U) + \rho(Y \cup U) + d(Y, U)$. Using (3), $\rho(Y \cup U) \geq \lambda - 1 = \rho(U)$. Thus to show the three desired equations it suffices to prove $\rho(Y \cap U) \geq \rho(Y)$.

If $\rho(Y) = \lambda - 1$ then (3) shows $\rho(Y \cap U) \geq \rho(Y)$ as desired. Suppose $\rho(Y) > \lambda - 1$. Since $Y \in \mathcal{F}$, (4) shows $\rho(Y) = \rho_G(Y) = \lambda$ and $\rho_A(Y) = 0$. Since A is an arborescence the latter implies $\rho_A(Y \cap U) = 0$. The definition of λ now implies $\rho(Y \cap U) \geq \lambda$ as desired.

To complete the proof it remains to show that $\rho(Z) = \lambda - 1$. Since $Z = \bigcup\{Y \cup U : Y \in YI \cup \{\emptyset\}\}$, repeated applications of the submodular identity, using (3) and the relations $\rho(Y \cup U) = \lambda - 1$, give the desired equation. □

The following algorithm constructs one spanning a-arborescence A for the packing.

procedure *grow*; **begin**
$A \leftarrow$ any edge leaving a;
while A is not spanning **do begin**
let e be an edge leaving $V(A)$ that does not enter any set $X \in \mathcal{F}$ with $\rho_{G-A}(X) = \lambda - 1$;
if $A \cup e$ satisfies (3) **then** add e to A **else begin**
$U \leftarrow$ the minimal set entered by e and having $\rho_{G-A}(U) = \lambda - 1$;
uncross $\mathcal{F} \cup \{U\}$; **end**; **end**; **end**;

To show this algorithm is well-defined, note there is a unique minimal set U (if $e = vw$, U is a $w\overline{a}$-set of minimum in-degree). Next we will prove that the edge e exists. For $X \in \mathcal{F} \cup \{V\}$

recall the definition of G_X from Section 2. Choose any minimal set $X \in \mathcal{F} \cup \{V\}$ where A contains at least one but not every vertex of G_X (strictly speaking we do not mean A here, but rather the arborescence corresponding to A in the contracted graph G_X).

Lemma 4.2. *G_X contains an edge leaving $V(A)$.*

Proof. Let S be the set of vertices of G_X not contained in A. A contains every vertex of G that is in $X - S$ or in a contracted vertex in $X - S$ (by the minimality of X). Thus it suffices to show that some edge of G goes from $X - S$ to S. This is clear for $X = V$ since $\lambda > 0$. For $X \neq V$ the fact that A enters X but not S shows that $\rho_{G-A}(S) \geq \lambda = \rho_{G-A}(X) + 1$. Thus $G - A$ contains an edge entering S but not X. This is the desired edge. □

It is easy to see that the edge of the lemma can be used for e in *grow*.

We will show below that *grow* actually halts, finding a spanning a-arborescence A. Assuming that, consider the following packing algorithm, *one-by-one_pack*: Initialize $\mathcal{F}$ to $\emptyset$. Execute *grow* to find a spanning arborescence A. Then add A to the packing (with multiplicity one) and replace G by $G - A$. If $\lambda(G) > 0$ then repeat the process.

Theorem 4.1. *Procedure* one-by-one_pack *finds an integral packing of $\lambda(G)$ arborescences.*

Proof. Assume temporarily that every iteration of *grow* either adds an edge to A or enlarges $\mathcal{F}$. The latter can occur at most $n - 2$ times. Thus each execution of *grow* eventually returns a spanning arborescence A. Adding A to the packing increases its multiplicity by 1 while deleting A from G decreases λ by 1 (by (3)). Furthermore deleting A preserves (4), since immediately before deleting each $X \in \mathcal{F}$ has $\rho_{G-A}(X) = \lambda - 1$. Thus it is clear that the desired packing of $\lambda(G)$ arborescences is eventually obtained.

It remains to prove the assumption. Suppose an iteration does not enlarge A, i.e., $A \cup e$ does not satisfy (3). The uncrossing operation adds a set Z to $\mathcal{F}$. Lemma 4.1 shows $\rho_{G-A}(Z) = \lambda - 1$, i.e., (4) holds. U is not a singleton (if $e = vw$, $\rho_{G-A}(w) \geq \lambda$). We claim that edge e enters Z. This shows $Z \notin \mathcal{F}$ before uncrossing (by the choice of e). Thus $\mathcal{F}$ is actually enlarged as desired.

To prove e enters Z we assume the contrary and derive a contradiction. Let $e = vw$. Since e enters U but not Z, $v \in Y - U$ for some $Y \in YI$. This implies $w \in Y \cap U$, since $e \in G - A$ and $d_{G-A}(Y, U) = 0$ (Lemma 4.1). Thus e enters $Y \cap U$. Furthermore $\rho_{G-A}(Y \cap U) = \lambda - 1$. (To see

this note that $\rho_A(Y) > 0$. Since $Y \in \mathcal{F}$ this implies $\rho_{G-A}(Y) = \lambda - 1$ by (4). Now Lemma 4.1 gives the desired equation.) The last two properties of $Y \cap U$ contradict the minimality of U. □

The theorem gives an independent proof of Edmonds' theorem [E72], i.e., any digraph with integral capacities has a packing of $\lambda(G)$ a-arborescences. Procedure *one-by-one_pack* can be efficiently implemented on a graph: We use the matroid approach of [G91] to check condition (3) and find the set U if (3) fails. This algorithm achieves the same time bound as [G91]: if G has kn edges and $\lambda(G) = k$, it finds a packing of k arborescences in time $O((kn)^2)$. In practice *one-by-one_pack* may be faster, since it may be easier to maintain $\mathcal{F}$ than to do the graph contraction operations of [G91].

The efficient integral packing algorithm of Section 5 uses a modified version of *grow*, that gains efficiency by not checking (3) for each edge e. For $e \in A$ let A/e denote the set of all edges added to A strictly before e. The following *fast_grow* procedure is called with A a spanning a-arborescence satisfying $\rho_A(X) \leq 1$ for every $X \in \mathcal{F}$. The procedure halts with A a spanning a-arborescence that satisfies (3).

procedure *fast_grow*;
while A does not satisfy (3) **do begin**
  $e \leftarrow$ the last edge added to A with A/e satisfying (3);
  $A \leftarrow A/e$;
  $U \leftarrow$ the minimal set entered by e having $\rho_{G-A}(U) = \lambda - 1$;
  uncross $\mathcal{F} \cup \{U\}$;
  repeat /* "grow loop" */
    add an edge to A that leaves $V(A)$
      and does not enter any set $X \in \mathcal{F}$ with $\rho_{G-A}(X) = \lambda - 1$;
  until A is spanning;
end;

In the first iteration of *fast_grow*, assume the edges of A have been added in an order consistent with the grow loop, i.e., for each e, A/e is an a-arborescence.

To show that *fast_grow* is correct first note that A/e is an arborescence (by the grow loop and the definition of /). Next observe that the grow loop constructs a spanning arborescence A. This follows from repeated application of Lemma 4.2. Finally observe that *fast_grow* is correct because it works exactly the same as *grow*. More precisely immediately after each uncrossing operation in

fast_grow, A and $\mathcal{F}$ are exactly the same as they are in an execution of *grow* that has constructed the arborescence $A/e \cup e$.

The integral packing algorithm of Section 5 calls *fast_grow* with an initial arborescence A that does not satisfy (3). Now we give some implementation details of *fast_grow* and also show that the total time for all such executions of *fast_grow* is

$$O(n^2 m \log(n^2/m) \log n).$$

We check that A satisfies (3) using the minimum cut algorithm of Hao and Orlin [HO]. This uses time $O(nm \log(n^2/m))$. If (3) fails we find the desired edge e using a binary search. The search maintains an interval of edges that were added consecutively to A and contain e. Each probe chooses an appropriate edge $f \in A$ and checks if A/f satisfies (3), again using the Hao-Orlin algorithm. This uses time $O(nm \log(n^2/m) \log n)$. The last execution of the Hao-Orlin algorithm finds the desired minimal set U.

The algorithm maintains the forest representing $\mathcal{F}$. This allows us to uncross $\mathcal{F} \cup \{U\}$ in time $O(n)$. Also we find each arborescence A in $O(m)$ time.

If an iteration of (the while-loop of) *fast_grow* starts with A not satisfying (3), the uncrossing operation enlarges $\mathcal{F}$. This follows from the proof of Theorem 4.1 and the correspondence between *fast_grow* and *grow*. Since each call to *fast_grow* starts with such an A, we conclude there are a total of $O(n)$ iterations in all executions of *fast_grow*. This implies the desired time bound.

5 Integral Packing

This section presents our algorithm to find a maximum integral packing, using at most $m + n - 2$ distinct arborescences.

Let $G = (V, E)$ be a digraph with a distinguished vertex a and an integral capacity function c. For a set of edges A, the *integral capacity of* A is the largest integral value $\alpha \leq \mu(A)$ having $\lambda(G - \alpha A) = \lambda(G) - \alpha$. The integral capacity equals the floor of the capacity defined in Section 2. Our algorithm packs arborescences A according to their integral capacity. With each such A of capacity $< \mu(A)$ we pack another arborescence (of multiplicity 1) that enlarges $\mathcal{F}$. The details are as follows.

Let $\mathcal{A}$ be a family of a-arborescences A, each with an associated integral multiplicity $\tau(A)$. $G - \mathcal{A}$ denotes the graph obtained by starting with G and deleting $\tau(A)$ copies of each $A \in \mathcal{A}$. $\mathcal{A}$ is *valid* if each $A \in \mathcal{A}$ is a spanning a-arborescence and for $\tau(\mathcal{A})$ the total multiplicity of $\mathcal{A}$,

$\lambda(G - \mathcal{A}) = \lambda(G) - \tau(\mathcal{A})$. For $A \in \mathcal{A}$, $\mathcal{A}/A$ denotes the family of all arborescences added to $\mathcal{A}$ strictly before A. The integral packing algorithm is as follows.

procedure *integral_pack*; **begin**
$\mathcal{F} \leftarrow \emptyset$;
while $\lambda(G) > 0$ **do begin**
$\mathcal{A} \leftarrow \emptyset$;
repeat
$A \leftarrow$ a maximal a-arborescence in $G - \mathcal{A}$ with $\rho_A(X) \leq 1$ for every $X \in \mathcal{F}$;
add A with multiplicity $\mu(A)$ to $\mathcal{A}$;
until A is not spanning or $|\mathcal{A}| = n$;
if $\mathcal{A}$ is valid **then** $G \leftarrow G - \mathcal{A}$
else begin
$B \leftarrow$ the last arborescence added to $\mathcal{A}$ with $\mathcal{A}/B$ valid;
$G \leftarrow G - \mathcal{A}/B$;
if B is spanning **then begin**
$\beta \leftarrow$ the integral capacity of B (in the current graph G);
$G \leftarrow G - \beta B$;
call *fast_grow* with a-arborescence B, to return a-arborescence A;
$G \leftarrow G - A$; **end**; **end**; **end**; **end**;

As usual assume that whenever we delete arborescences from G we add these arborescences to the packing, with the corresponding multiplicities. Also note that procedure *fast_grow* maintains the laminar family $\mathcal{F}$. Finally we remark that B can be the first arborescence of $\mathcal{A}$, and it can have capacity 0.

We now show that *integral_pack* finds an integral packing of total multiplicity $\lambda(G)$. Whenever we delete a valid family of arborescences from G, the packing is enlarged correctly (i.e., the decrease in $\lambda(G)$ equals the increase in total multiplicity of the packing). Also every set $X \in \mathcal{F}$ always satisfies (4) of Section 4. In particular throughout the execution of *integral_pack* except during the execution of *fast_grow*, $\rho_G(X) = \lambda(G)$. This condition is preserved when we delete a valid family of arborescences (since any arborescence in the family necessarily enters X only once). These remarks show that *integral_pack* works correctly when $\mathcal{A}$ is valid.

Suppose $\mathcal{A}$ is invalid. Procedure *integral_pack* works correctly when $\mathcal{A}/B$ is deleted. Consider the case when arborescence B is not spanning. We claim that the current graph has $\lambda(G) = 0$.

In proof, when B was added to $\mathcal{A}$ (in the repeat-loop) the graph $G - \mathcal{A}$ was precisely the current graph G. In general during the repeat-loop constructing $\mathcal{A}$, the arborescence A is spanning if $\mathcal{A}$ is valid and $\lambda(G-\mathcal{A}) > 0$. This follows from Lemma 4.2. The claim now follows. It is easy to see that *integral_pack* halts with the desired packing in this case.

Finally consider the case when B is spanning. The algorithm works correctly when βB is deleted. After the deletion each edge of B still has positive capacity and B is a spanning arborescence with integral capacity 0. Thus *fast_grow* is called with an arborescence as described in Section 4. (In particular B does not satisfy (3). Also as noted above, (4) holds for all sets in $\mathcal{F}$.) Hence *fast_grow* returns an arborescence A that is valid and can be added to the packing.

Since each iteration of *integral_pack* adds at least one arborescence to the packing, the procedure eventually halts with a packing of total multiplicity $\lambda(G)$.

Theorem 5.1. *Procedure* integral_pack *finds an integral packing with total multiplicity* $\lambda(G)$*, containing at most* $m + n - 2$ *distinct arborescences.*

Proof. Let $\mathcal{A}$ be the packing obtained for the original input graph. It remains only to check the number of distinct arborescences in $\mathcal{A}$. At most $m-n+2$ arborescences A have multiplicity $\mu(A)$ in $\mathcal{A}$. This follows since each such A voids an edge, and the last arborescence added to $\mathcal{A}$ contains $n-1$ edges. The remaining arborescences of $\mathcal{A}$ come in pairs B, A that are the input and output of an execution of *fast_grow*. Section 4 shows that each execution of *fast_grow* does an uncrossing operation that enlarges $\mathcal{F}$. Hence there are at most $n-2$ such calls. Thus $|\mathcal{A}| \leq (m-n+2)+2(n-2) = m+n-2$. □

We conclude by giving the remaining implementation details of *integral_pack* and computing the time bound.

Corollary 5.1. *Procedure* integral_pack *runs in time* $O(\min\{n, \log(nN)\}n^2 m \log(n^2/m))$.

Proof. Each iteration starts by finding $\leq n$ arborescences in time $O(nm)$. Consider the iterations having $\mathcal{A}$ valid. Validity is checked using the Hao-Orlin minimum cut algorithm [HO]. Thus the time for such an iteration is $O(nm \log(n^2/m))$. An iteration with $\mathcal{A}$ valid has $|\mathcal{A}| = n$. (If $|\mathcal{A}| < n$ the last arborescence of $\mathcal{A}$ is not spanning and $\mathcal{A}$ is invalid.) We conclude that there are $\leq m/n \leq n$ iterations having $\mathcal{A}$ valid. The total time for these iterations is $O(n^2 m \log(n^2/m))$.

Next consider the iterations having $\mathcal{A}$ invalid. The arborescence B is found by a binary search in time $O(nm \log(n^2/m) \log n)$. Lemma 3.1 shows that the integral capacity of B is found in time $O(\min\{n, \log(nN)\}\ nm \log(n^2/m))$. As noted in the analysis of *fast_grow*, at most n iterations have $\mathcal{A}$ invalid. (This includes at most one iteration where *fast_grow* is not called, since as noted above such an iteration, where B is not spanning, is the last.) This contributes total time equal to the desired bound. Also the total time for all executions of *fast_grow* (computed at the end of Section 4) is within the desired bound. □

6 Applications of Packing

This section applies our packing algorithms to related problems, specifically, covering digraphs by branchings, and packing and covering in undirected graphs.

Sections 6 and 7 use various graphs having a parametrized capacity function. Specifically the given graph G has a capacity function c_p, where the parameter p can take on any real value that is at least some fixed lower bound (e.g., 0). Unless otherwise specified we use G to denote the given graph with capacity function c_p, where the value of p is determined by the context. Similarly the function ρ denotes ρ_G where G uses capacity function c_p.

We first consider the problem of covering a digraph by branchings. More precisely let G be a digraph with capacity function c. We use terminology analogous to packing. The problem is to find a family of branchings, each with an associated multiplicity, such that each edge e occurs in branchings of total multiplicity equal to $c(e)$ and the family has the minimum possible total multiplicity. This minimum multiplicity is called the *branching covering number*. We consider both fractional and integral covering; correspondingly we have the fractional and integral branching covering numbers.

We characterize the branching covering number as follows. Form the digraph $\overline{G}$ from G by adding a new vertex a and an edge av for every vertex $v \in V$. Let $d = \max\{\rho_G(v) : v \in V\}$. For $p \geq d$ define a parameterized capacity function c_p on $\overline{G}$ as follows: If edge e leaves vertex a, say $e = av$, then $c_p(e) = p - \rho_G(v)$; otherwise $c_p(e) = c(e)$. The expression $\lambda(\overline{G})$ is interpreted according to two of our conventions: As stated above, $\overline{G}$ uses the capacity function c_p. As stated in Section 1, $\lambda(\overline{G})$ is defined using a as the root vertex. Note that $\lambda(\overline{G}) \leq p$ since each vertex $\neq a$ has in-degree p in $\overline{G}$.

Lemma 6.1. *The (integral or fractional) branching covering number of G is the smallest (integral or fractional) value p such that $\lambda(\overline{G}) = p$.*

Proof. Let b be the (integral or fractional) branching covering number of G. Let p be the smallest (integral or fractional) value as specified in the lemma. We prove $b = p$ in two steps.

We first show $b \geq p$. In this argument fix c_b as the capacity function on $\overline{G}$. We convert a covering of G having multiplicity b to a packing of a-arboresences in $\overline{G}$ having total multiplicity b, as follows: Add edge av to each branching that has v as a root, for each $v \in V$. This gives a packing of arborescences, in which each edge av has multiplicity $b - \rho_G(v) = c_b(av)$. Hence this is a valid packing in $\overline{G}$. (Note this argument also ensures the existence of the quantity p.)

Next we show $p \geq b$. In this argument fix c_p as the capacity function on $\overline{G}$. We convert a packing $\mathcal{A}$ of a-arborescences in $\overline{G}$ having total multiplicity p to a covering of G having total multiplicity p, as follows: Delete all occurrences of edges av from $\mathcal{A}$. Note that $\mathcal{A}$ contains every edge of $\overline{G}$ with multiplicity equal to its capacity (since as noted each vertex $\neq a$ has in-degree p). Thus the resulting family of branchings is a cover of G. □

We use the parametric s-set algorithm of [G95] to calculate the quantity of the lemma. This algorithm is also used in Section 7. We now summarize the parametric s-set problem and all properties of the algorithm needed in Sections 6 and 7.

The *parametric s-set problem* is defined by a digraph G with distinguished vertex a, capacity function c_p parameterized by $p \geq 0$, and target function $\tau(p)$. The problem is to find the smallest value of p such that using capacity function c_p, $\lambda(G) \geq \tau(p)$ (this value is infinite if no such p exists).

We make several assumptions to ensure this problem is well-defined and tractable: Any $\overline{a}$-set S has a value p_S (possibly infinite) such that $\rho(S) \geq \tau(p)$ exactly when $p \geq p_S$. Furthermore given S we can compute p_S in time $O(m)$. Also given p and an edge e we can compute $c_p(e)$ in $O(1)$ time. Finally we assume monotonicity properties for the capacities: Any edge e leaving vertex a has $c_p(e)$ a nondecreasing function of p, and any other edge e has $c_p(e)$ constant. For arbitrary real-valued capacities c_p, the algorithm of [G95] solves the parametric s-set problem in time $O(nm \log(n^2/m))$ and space $O(m)$.

The parametric s-set algorithm starts with $p = 0$. Then it repeatedly finds a set S with $\rho(S) < \tau(p)$ and increases p to p_S (so $\rho(S) \geq \tau(p)$) until $\lambda(G) \geq \tau(p)$. (Each time we change p this

changes the capacity function c_p.) The values p_S and $c_p(e)$ are computed by a subroutine – they can be chosen as the algorithm executes.

We use the parametric s-set algorithm to calculate the branching covering number as follows. The parametric s-set problem is defined as in Lemma 6.1, i.e., the graph is $\overline{G}$ with distinguished vertex a, capacity function c_p and target function $\tau(p) = p$. (Our parameter p has lower bound d rather than 0 but this is inconsequential.) Lemma 6.1 shows the branching covering number equals the answer to the parametric s-set problem.

Note that the values p_S are defined and easy to calculate: Any set S having $\rho(S) \geq d$ for capacity function c_d has $p_S = d$ (for example this includes any singleton set S). Any other set S has $p_S = d + (d - \rho(S))/(|S| - 1)$, where ρ uses capacity function c_d. (For fractional covering this formula follows easily from the definitions. For integral covering use the ceiling of this value.) The monotonicity property for c_p is obvious.

Theorem 6.1. *The (integral or fractional) branching covering number of a digraph can be found in time $O(nm \log(n^2/m))$. A corresponding fractional covering can be found in the time bound of Corollary 2.1, using at most $m + n$ distinct branchings. For integral covering the time is the same as Corollary 5.1, using at most $m + 2n - 1$ distinct branchings.*

Proof. We have shown how to calculate the branching covering number b in the desired time. A corresponding minimum cover can be found by executing our packing algorithm on $\overline{G}$ with capacities c_b. The bounds on the number of distinct branchings in the cover follow from Theorems 2.1 and 5.1. □

We turn to problems on undirected graphs G, specifically, covering G by forests and packing spanning trees in G. [G95] shows that the integral and fractional versions of both of these problems reduce to packing a-arborescences in a digraph D. (D is called the reverse of the equivalent graph, EG_k^R, in [G95].) D is constructed as follows. We begin with a subgraph SG of G. For the covering problem $SG = G$. For the packing problem [G95] constructs SG; it is the union of all spanning trees of the desired packing (with corresponding capacity function). Form D by replacing each edge vw of SG by directed edges vw and wv, and adding a new vertex a and an edge av for each $v \in V$. The capacity function on D is derived from a maximum flow computation. For the covering problem let k be the total multiplicity of a minimum covering by forests of G. For the packing problem let k be the total multiplicity of a maximum packing of spanning trees of G. Then D has a packing

of a-arborescences of total multiplicity k. Deleting a from these arborescences and ignoring edge directions gives the desired covering or packing on G.

We make three remarks on this construction. First consider fractional covering and packing on G. [G95] assumes rational capacities to prove that a packing on D gives the desired covering or packing on G. Our proof of Edmonds' theorem for real capacities shows (a minor modification of) the argument of [G95] is valid for real capacities. Next consider integral covering and packing on G. In this case D has integral capacities, and we use an integral packing on D. Finally note that [G95] shows D can be constructed in time $O(nm\log(n^2/m))$ for covering and $O(n^2m\log(n^2/m))$ for packing.

We find the desired covering or packing on G using our packing algorithm on D. Since D has at most $2m+n$ edges we obtain a minimum covering by forests using at most $2m+n$ $(2m+2n-1)$ distinct forests for fractional (integral) covering, by Theorem 2.1 (5.1). The same two bounds hold for the number of distinct spanning trees in a maximum packing. We now reduce these bounds by reducing the number of edges in D.

First we review the construction of D from [G95] in more detail. Let $SG = (V,E)$ with capacity function c_{SG}. Let s and a be new vertices not in V. Define a digraph G^* to have vertex set $\{s,a\} \cup V \cup E$ and edge set $\{se : e \in E\} \cup \{ev, ew : e = vw \in E\} \cup \{va : v \in V\}$. Define a capacity function c on G^* by the identities $c(se) = c_{SG}(e)$, $c(ev) = c(ew) = \infty$ and $c(va) = k$. As above, k denotes the (integral or fractional) covering or packing number of G.

Next take f to be a maximum flow from s to a on G^*. Use an integral flow for integral covering or packing. ([G95] shows that f saturates all edges leaving s.) Form the residual graph of f. Delete vertex s and delete all edges leaving a. Then do the following for each $e = xy \in E$: delete the vertex e and replace it by edges xy and yx of capacities $f(ex)$ and $f(ey)$ respectively. Digraph D is the reverse of this graph.

An edge of D not incident to a corresponds to an edge ev of G^* with positive flow. We achieve a small number of such edges as follows. Call a flow on G^* *acyclic* if there is no undirected cycle of edges ev where each edge has positive flow. G^* has a maximum flow that is acyclic. In proof, any edge ev has infinite capacity. Hence for any undirected cycle of edges ev carrying flow, we can push flow around the cycle in a direction that voids an edge. Doing this repeatedly gives the desired acyclic flow. This proof can be implemented efficiently using the dynamic tree data structure. This allows us to convert any maximum flow on G^* to a maximum acyclic flow in time $O(m\log n)$. This result is due to Sleator and Tarjan [ST].

Theorem 6.2. *A minimum fractional covering of a capacitated undirected graph by forests can be found in the time bound of Corollary 2.1, using at most $m + 2n - 2$ distinct forests. For integral covering the time is the same as Corollary 5.1, using at most $m + 3n - 2$ distinct forests.*

Proof. We need only prove the bounds on the number of distinct forests. In an acyclic flow at most $m + n - 1$ edges ev have positive flow. Using a maximum acyclic flow in G^* gives a graph D with $\leq (m + n - 1) + n = m + 2n - 1$ edges. The desired bound for integral covering now follows from Theorem 5.1.

Theorem 2.1 shows we use at most $m + 2n - 1$ forests for fractional covering. This number decreases by 1 if we initialize *fractional_pack* so the laminar family $\mathcal{F}$ contains a set S. Such a set is available from the algorithm of [G95] that constructs D. (This follows from the arboricity algorithm of [G95], Section 5. To show this we use the terminology of [G95]. Assuming the graph G has some edge of positive capacity, the fractional arboricity is strictly greater than the fractional density, $\Gamma > d$. This implies the second step of the arboricity algorithm, which executes the parametric s-set algorithm, increases the parameter p at least once. The last such increase makes some nonsingleton $\overline{a}$-set of vertices S have $\rho_D(S) = \Gamma$. In our notation Γ is k. Hence S can be added to $\mathcal{F}$.) □

The above discussion also holds for packing. But we can further reduce the number of distinct trees in the packing.

Theorem 6.3. *A maximum fractional packing of spanning trees in a capacitated undirected graph can be found in the time bound of Corollary 2.1, using at most $m + n - 2$ distinct trees. For integral packing the time is the same as Corollary 5.1, using at most $m + 2n - 4$ distinct trees.*

Proof. We need only prove the bounds on the number of distinct spanning trees. To do this we modify the construction of D as follows.

Fix an arbitrary vertex b. G^* has a maximum flow with edge ba void. This is easy to show using the max flow min cut theorem ([G95] gives similar arguments). Here is an alternate proof: By definition SG has a packing $\mathcal{T}$ of spanning trees of total multiplicity k. Root every tree of $\mathcal{T}$ at vertex b. For any edge $e = vw$ define $f(ew)$ ($f(ev)$) to be the total multiplicity of all arborescences that contain the directed edge vw (wv). Extending f in the natural way gives a flow on G^* that saturates all edges leaving s (i.e., f is maximum). Observe that $f(ba) = 0$ (no flow enters b since it

is the root of every tree of $\mathcal{T}$) and any other edge va has $f(va) = k$ (since v is not the root of any tree of $\mathcal{T}$).

Now make the above flow f acyclic. At most $m+n-2$ edges ev have positive flow (recall that no edge eb has positive flow). In the residual graph of this flow only one edge ba enters a, since any other edge va is saturated. Thus we get a graph D with $\leq m+n-1$ edges. Observe that in the packing of a-arborescences of multiplicity k in D, each arborescence contains edge ab. So we can delete this edge from D, and pack b-arborescences in $D - \{ab\}$ to get the desired packing of spanning trees in G.

In summary we pack arborescences in a digraph having $m+n-2$ edges. Now the desired bounds follow from Theorems 2.1 and 5.1. □

7 Root-constrained Packing

Consider a packing of spanning arborescences where each arborescence is rooted at an arbitrary vertex v. Let $r(v)$ denote the total multiplicity of all arborescences rooted at v. In a *maximum root-constrained packing* we seek a packing of spanning arborescences with the largest total multiplicity subject to the constraint that each value $r(v)$ lies in a given real interval $[\ell(v), u(v)]$. We allow $u(v)$ to be infinite (which is equivalent to letting $u(v)$ equal the out-degree of v). The case $\ell(v) = u(v) = 0$ means v is never a root. This section gives an algorithm that computes the above maximum multiplicity, as well as each value $r(v)$ in the desired packing. It is then a simple matter to find the desired packing.

Our algorithm is based on minimax formulas due to Frank [F78] (which treats the case of upper bounds only) and Mao-Cheng [Mao] (which treats the case of lower bounds only). In fact our algorithm is similar to the proof of [F78]. Frank and Tardos [FT] showed that the two minimax formulas can be combined to give the following theorem: A digraph has k edge-disjoint arborescences such that $\ell(v) \leq r(v) \leq u(v)$ for every vertex v iff these two conditions hold:

$$\rho_G(S) + u(S) \geq k \quad \text{for every nonempty set } S \subseteq V \tag{5a}$$

$$\sum\{\rho_G(S_i) : 1 \leq i \leq t\} \geq \ell(S_0) + k(t-1) \quad \text{for every partition of } V \text{ into sets } S_i,\ 0 \leq i \leq t, \text{ where } t \geq 0 \text{ and } S_i \text{ is nonempty for } 1 \leq i \leq t \tag{5b}$$

Here and throughout this section if S is a set of vertices then $u(S)$ denotes $\sum\{u(v) : v \in S\}$, and similarly for $\ell(S)$. Our development gives another proof of this result, for both the fractional and integral cases. We discuss the fractional case as the default, and note the minor modifications needed for the integral case.

The algorithm uses the digraph $\overline{G}$ which is formed from G by adding a new vertex a. Every vertex $v \in V$ has a new edge av of capacity $u(v)$. (This capacity, which we always denote $u(v)$, gets decreased during the algorithm.) We use two in-degree functions: ρ and $\overline{\rho}$ denote ρ_G and $\rho_{\overline{G}}$ respectively ($\overline{\rho}$ changes as $\overline{G}$ changes.) Observe that $\lambda(\overline{G}) = u(V)$ iff G has a packing of total multiplicity $u(V)$, with each vertex v the root of exactly $u(v)$ arborescences. (As always $\lambda(\overline{G})$ is defined using a as the root.) Thus our plan is to decrease u values until eventually the desired packing exists.

Inequalities (5a–b) will guide the decrease, so let us verify that they are necessary conditions for the desired root-constrained packing: Suppose G has a root-constrained packing of multiplicity k. (5a) holds since $\lambda(\overline{G}) \geq k$. To show (5b) consider a spanning arborescence A of G, rooted at a vertex v. Any set S_i not containing v has $\rho_A(S_i) \geq 1$. Thus A contributes $t-1$ to the left-hand side of (5b) if $v \notin S_0$ and t if $v \in S_0$. (5b) follows.

Our algorithm maintains a variable k that decreases to the maximum multiplicity of a root-constrained packing (if such exists). The values $u(v)$ also decrease from their given values but not below $\ell(v)$. The algorithm returns the final values of k and u, where k is the maximum multiplicity of a root-constrained packing $\mathcal{A}$, and each vertex v is the root of arborescences of total multiplicity $u(v)$ in $\mathcal{A}$.

procedure *find_roots*; **begin**
$k \leftarrow \lambda(\overline{G})$;
while true do begin
decrease values $u(v), v \in V$ as much as possible
maintaining $u(v) \geq \ell(v)$ and $\lambda(\overline{G}) \geq k$; /* "decrease step" */
if $u(V) = k$ **then return** k, u;
$\mathcal{T} \leftarrow \{S : S$ is a maximal $\overline{a}$-set with $\overline{\rho}(S) = k\}$; $t \leftarrow |\mathcal{T}|$;
if $t \leq 1$ **then return** "no root-constrained packing exists";
$S_0 \leftarrow V - \bigcup\{S : S \in \mathcal{T}\}$;
$k \leftarrow (\sum\{\rho(S) : S \in \mathcal{T}\} - \ell(S_0))/(t-1)$;
/* for integral packing set k to the floor of this value */
end; **end**;

Now we prove that *find_roots* returns the correct answer. The argument also proves that (5a–b) are necessary and sufficient conditions for root-constrained packing (in particular when no root-constrained packing exists it exhibits a violated condition).

Lemma 7.1. *Procedure* find_roots *is correct.*

Proof. We begin by showing that k is nonincreasing. Note this implies the algorithm maintains the invariant $\lambda(\overline{G}) \geq k$. Let an iteration start with the value k and end with the value k'. We will show $k \geq k'$.

The definitions imply $\sum\{\rho(S) : S \in \mathcal{T}\} = \sum\{\overline{\rho}(S) - u(S) : S \in \mathcal{T}\} = kt - u(V) + u(S_0)$. Since we can assume that $\lambda(\overline{G}) \geq k$ after the decrease step,

$$u(V) = \overline{\rho}(V) \geq k. \tag{6}$$

If $u(v) > \ell(v)$ then v is in a set of $\mathcal{T}$. Thus $u(S_0) = \ell(S_0)$. Hence

$$\sum\{\rho(S) : S \in \mathcal{T}\} - \ell(S_0) = kt - u(V) \leq k(t-1). \tag{7}$$

Thus $k \geq k'$.

Next we show that $k = k'$ implies the next iteration halts. The assumed equality implies equality holds in (7) (for the integral case note that $k'(t-1)$ is at most the left-hand side). Thus equality holds in (6). Hence $u(V) = k = k'$ so the next iteration of *find_roots* halts (returning k, u).

We conclude that *find_roots* eventually halts. In proof suppose the current iteration computes a family of sets $\mathcal{T}$ that was computed in some previous iteration. The current iteration computes k' which is the same as the value computed in that previous iteration. Since k is nonincreasing we conclude that $k = k'$. Thus the next iteration halts.

It remains to show that *find_roots* returns the correct answer. First note that for the first value of k there is a set S satisfying (5a) with equality (using the given values u). For subsequent values of k there is a family $\mathcal{T}$ satisfying (5b) with equality. (In the integral case equality need not hold but k is the largest integer satisfying (5b).)

If *find_roots* returns with $u(V) = k$ the definition of $\overline{G}$ shows G has a root-constrained packing with total multiplicity k. Furthermore k is as large as possible, by the equality in (5a) or (5b) noted above.

Next suppose *find_roots* returns with $t \leq 1$. If $t = 0$ then no $\overline{a}$-set S has $\overline{\rho}(S) = k$. Hence the decrease step shows every vertex has $u(v) = \ell(v)$. Thus $\ell(V) = u(V) > k$ (recall $\lambda(\overline{G}) \geq k$). Thus the equality in (5a) or (5b) for k implies $\ell(V)$ is too large for a root-constrained packing.

The remaining possibility is $t = 1$, i.e., $\mathcal{T}$ consists of the single set S. In this case $u(V) > k = \overline{\rho}(S)$. Subtracting $u(S)$ shows $u(S_0) = \ell(S_0) > \rho(S)$. This violates (5b) (with $t = 1$) so no root-constrained packing exists. □

We now prove the key fact for the efficiency of *find_roots*. Define $\mathcal{F}$ to be the family of all sets S such that at some point in the algorithm S is a maximal $\overline{a}$-set with $\overline{\rho}(S) = k$. Any set in any $\mathcal{T}$ family belongs to $\mathcal{F}$. But $\mathcal{F}$ also contains sets S that satisfy the definition at some point during the execution of the decrease step but not at the end of that step. This will be important when we implement the decrease step.

Lemma 7.2. *$\mathcal{F}$ is a laminar family. Procedure* find_roots *performs $O(n)$ iterations.*

Proof. Call a set $S \subseteq V$ *extreme* if any nonempty subset $T \subseteq S$ has $\overline{\rho}(T) \geq \overline{\rho}(S)$. (This is a slight abuse of terminology since the usual definition of extreme set requires strict inequality.) Observe that at any point in the algorithm every set $S \in \mathcal{F}$ is extreme. This follows because S is extreme when it enters $\mathcal{F}$, since at that time any set $T \subseteq S$ has $\overline{\rho}(T) \geq k = \overline{\rho}(S)$. After that S remains extreme, since the decrease step preserves extreme sets.

Now we show that $\mathcal{F}$ is always laminar. After k is initialized $\mathcal{F}$ is a family of disjoint sets (the maximal $\overline{a}$-sets of in-degree $k = \lambda(\overline{G})$). Suppose a set S gets added to $\mathcal{F}$, i.e., S is a newly-created maximal $\overline{a}$-set with $\overline{\rho}(S) = k$. Take any set $T \in \mathcal{F}$ that is not disjoint from S. The submodular identity $\overline{\rho}(S) + \overline{\rho}(T) \geq \overline{\rho}(S \cap T) + \overline{\rho}(S \cup T)$ with $\overline{\rho}(S \cap T) \geq \overline{\rho}(T)$ (since T is currently an extreme set) shows $k = \overline{\rho}(S) \geq \overline{\rho}(S \cup T)$. Hence $k = \overline{\rho}(S \cup T)$ and by maximality $S = S \cup T$, i.e., $T \subseteq S$ as desired.

Next we observe two facts about sets of $\mathcal{T}$. First, a set $S \in \mathcal{T}$ is the union of maximal sets of $\mathcal{F}$ and vertices not in a set of $\mathcal{F}$. In proof suppose S is not disjoint from $T \in \mathcal{F}$. If $S \subseteq T$ then $\overline{\rho}(T) = k$ since T is extreme. Hence maximality implies $S = T$.

Second note that a given set S can be in $\mathcal{T}$ for a number of iterations. S drops out of $\mathcal{T}$ when it is no longer maximal in $\mathcal{F}$ or it achieves $u(S) = \ell(S)$. In both cases S never returns to $\mathcal{T}$.

Now we can prove that *find_roots* has $O(n)$ iterations. $O(n)$ iterations add a set to $\mathcal{F}$ (since $\mathcal{F}$ is laminar). Consider any other iteration of *find_roots* that is not the last or next-to-last. As shown in Lemma 7.1 the iteration changes the family $\mathcal{T}$. Thus some set of $\mathcal{F}$ drops out of $\mathcal{T}$. Since $|\mathcal{F}| = O(n)$ this occurs $O(n)$ times. □

Now we present the detailed implementation of *find_roots*, the main part of which is the decrease step. We implement the decrease step in a number of iterations. The net result of any iteration is that values $u(v)$ get decreased, maintaining $u(v) \geq \ell(v)$ and $\lambda(\overline{G}) \geq k$, and a new set S with $\overline{\rho}(S) = k$ gets created. The parametric s-set algorithm is used to accomplish this. (During the

execution of the parametric s-set algorithm values $u(v)$ get increased, but this does not invalidate the above description of an iteration.) The details of an iteration of the decrease step are as follows.

Each iteration of the decrease step starts by setting $u_0(v)$ to $u(v)$ for each vertex v. Then compute the family $\mathcal{T}$ using the Hao-Orlin algorithm. For each $S \in \mathcal{T}$ decrease values $u(v), v \in S$ as much as possible maintaining $u(v) \geq \ell(v)$ and $\overline{\rho}(S) \geq k$ (subject to these constraints, the choice of vertices to decrease is arbitrary). If now $\lambda(\overline{G}) \geq k$ the decrease step is complete (test this condition using the Hao-Orlin algorithm).

On the other hand if $\lambda(\overline{G}) < k$ we change $u(v)$ values by solving a parametric s-set problem. The problem is on graph $\overline{G}$ with distinguished vertex a. The target function $\tau(p)$ equals the constant k. The capacity function c_p has $c_p(e)$ nondecreasing for edges leaving a and constant otherwise. The exact values of c_p get specified during the execution of the parametric s-set algorithm. It will be apparent from the description that c_p satisfies the required properties (each value p_S exists and can be computed in $O(m)$ time, and capacities can be computed in $O(1)$ time). Recall the description of the parametric s-set algorithm given in Section 6.

We execute the parametric s-set algorithm, starting with all edges at their current capacity. Suppose this algorithm finds a set S with in-degree $\overline{\rho}(S)$ less than the target k. Calculate a new capacity function that achieves $\overline{\rho}(S) = k$ as follows (this capacity function corresponds to parameter value p_S): Increase values $u(v), v \in S$, maintaining $u(v) \leq u_0(v)$, until $\overline{\rho}(S) = k$. (This can be done since the values $u_0(v)$ make $\lambda(\overline{G}) \geq k$.) The parametric s-set algorithm repeats this process as necessary. It eventually returns with each new value $u(v) \leq u_0(v)$, $\lambda(\overline{G}) = k$ and some set S having $\overline{\rho}(S) = k$ but $\overline{\rho}(S) < k$ immediately after u values were decreased (in the current iteration of the decrease step). This completes the iteration of the decrease step.

It is easy to implement the rest of *find_roots*. Use the Hao-Orlin algorithm to compute $\mathcal{T}$.

Theorem 7.1. *Procedure* find_roots *computes the maximum total multiplicity of a fractional or integral root-constrained packing, along with the multiplicity of each root vertex, in time* $O(n^2 m \log(n^2/m))$. *A corresponding fractional packing is found in the time bound of Corollary 2.1, using at most* $m + n - 1$ *distinct arborescences. For integral packing the time is the same as Corollary 5.1, using at most* $m + 2n - 3$ *distinct arborescences.*

Proof. First consider the efficiency of *find_roots*. Each execution of the parametric s-set algorithm uses time $O(nm \log(n^2/m)$ [G95]. Each such execution enlarges the family $\mathcal{F}$. Lemma 7.2 implies $\mathcal{F}$ gets enlarged $O(n)$ times. Thus the parametric s-set algorithm is executed $O(n)$ times. The

remaining time for *find_roots* amounts to $O(1)$ executions of the Hao-Orlin algorithm per iteration. Lemma 7.2 shows there are $O(n)$ iterations. The desired time bound follows.

We find the desired root-constrained packing by executing our packing algorithm on $\overline{G}$. The time is given by Corollary 2.1 or 5.1.

We finish the proof by bounding the number of distinct arborescences in the packing. We initialize our packing algorithms so the laminar family $\mathcal{F}$ contains the set V. This is permissible since $u(V) = k = \lambda(\overline{G})$. This initialization decreases the number of arborescences by 1 for fractional packing and 2 for integral packing. Now Theorems 2.1 and 5.1 give the desired bounds. (Note that if the given root constraints allow only r vertices to be roots, the bounds become $m + r - 1$ and $m + n + r - 3$ respectively.) □

Acknowledgments The first author wishes to pay tribute to Gene Lawler, who provided help and inspiration for many years.

References

[E65] J. Edmonds: Minimum partition of a matroid into independent subsets. J. Res. National Bureau of Standards **69B**, 1965, pp. 67–72

[E72] J. Edmonds: Edge-disjoint branchings. In: Combinatorial Algorithms, R. Rustin, Ed., Algorithmics Press, New York, 1972, pp. 91–96

[F78] A. Frank: On disjoint trees and arborescences. In: Algebraic Methods in Graph Theory, L. Lovász and V.T. Sós, Eds., Colloq. Math. Soc. János Bolyai **18**, North-Holland, New York, 1978, pp. 159–169

[F79] A. Frank: Kernel systems of directed graphs. Acta Sci. Math. **41**, 1979, pp. 63–76

[FT] A. Frank, É. Tardos: Generalized polymatroids and submodular flows. Math. Programming, 42, 1988, pp. 489–563

[G91] H.N. Gabow: A matroid approach to finding edge connectivity and packing arborescences. Proc. 23rd Annual ACM Symp. on Theory of Comp., 1991, pp. 112–122; J. CSS, to appear

[G95] H.N. Gabow: Algorithms for graphic polymatroids and parametric s-sets. Proc. 6th Annual ACM-SIAM Symp. on Disc. Algorithms, 1995, pp. 88–97

[GLS] M. Grötschel, L. Lovász, A. Schrijver: Geometric Algorithms and Combinatorial Optimization, Springer-Verlag, New York, 1988

[GW] H.N. Gabow, H.H. Westermann: Forests, frames and games: Algorithms for matroid sums and applications. Algorithmica **7**, 1992, pp. 465–497

[HO] J. Hao, J.B. Orlin: A faster algorithm for finding the minimum cut in a directed graph. J. Algorithms **17**, 3, 1994, pp. 424–446

[I] H. Imai: Network-flow algorithms for lower-truncated transversal polymatroids. J. Op. Res. Soc. of Japan **26**, 3, 1983, pp. 186–210

[L] L. Lovász: On two minimax theorems in graph theory. J. Comb. Theory, B, **21**, 1976, pp. 96–103

[Mad] W. Mader: On n-edge-connected digraphs. Annals Discr. Math. **17**, 1983, pp. 439–441

[Mao] C. Mao-cheng: Arc-disjoint arborescences of digraphs. J. Graph Theory **7**, 1983, pp. 235–240

[PW] M.W. Padberg, L.A. Wolsey: Fractional covers for forests and matchings. Math. Programming **29**, 1984, pp. 1–14

[R] T. Radzik: Newton's method for fractional combinatorial optimization. Proc. 33rd Annual Symp. on Found. of Comp. Sci., 1992, pp. 659–669

[RT] J. Roskind, R.E. Tarjan: A note on finding minimum-cost edge-disjoint spanning trees. Math. Op. Res. **10**, 4, 1985, pp. 701–708

[S] Y. Shiloach: Edge-disjoint branchings in directed multigraphs. Inf. Proc. Letters **8**, 2, 1979, pp. 24–27

[ST] D.D. Sleator, R.E. Tarjan: A data structure for dynamic trees. J. Comp. and System Sci. **26**, 1983, pp. 362–391

[Ta] R.E. Tarjan: A good algorithm for edge-disjoint branching. Inf. Proc. Letters **3**, 2, 1974, pp. 51–53

[Tro] L.E. Trotter, Jr.: Discrete packing and covering. In: Combinatorial Optimization: Annotated Bibliographies, M. O'hEigeartaigh, J.K. Lenstra and A.H.G. Rinnooy Kan, Eds., John Wiley, New York, 1985, pp. 21–31

[Tru91] V.A. Trubin: Strength and reinforcement of a network and tree packing. Kibernetika **2**, 1991, pp. 67–75

[Tru93] V.A. Trubin: Strength of a graph and packing of trees and branchings. Kibernetika i Sistemnyi Analiz **3**, 1993, pp. 94–99

[TL] P. Tong, E.L. Lawler: A faster algorithm for finding edge-disjoint branchings. Inf. Proc. Letters **17**, 2, 1983, pp. 73–76